README

Andrea Pierré

December 18, 2018

Contents

1 Design decisions

1.1 Data

For consistency reasons since *Node.js* was an imposed choice for the back-end, and to not impose to my reviewer to install another language, I would have chosen to load the data in *Javascript*. But since I was allowed to use Docker this was not a problem anymore. So since time was limited, I choose Python to get the data in the database as it was the language I was more confident with.

The *MySQL* official Docker image was more than 100MB, I thought it was overkill for a simple application like this one, so I eliminated *MySQL*. I surprisingly found a lean *alpine* version of *PostgreSQL* which was less than 30MB, so I hesitated between *PostgreSQL* and *SQLite*. At the end I chose to go with *PostgreSQL* because it was simpler to use with Docker. Without Docker I would have chosen to go with *SQLite*. I also chose to go with the *SQLAlchemy* ORM in case I had some problem down the road so that it

would be easy to switch to another database in case (and also because I wanted to learn it).

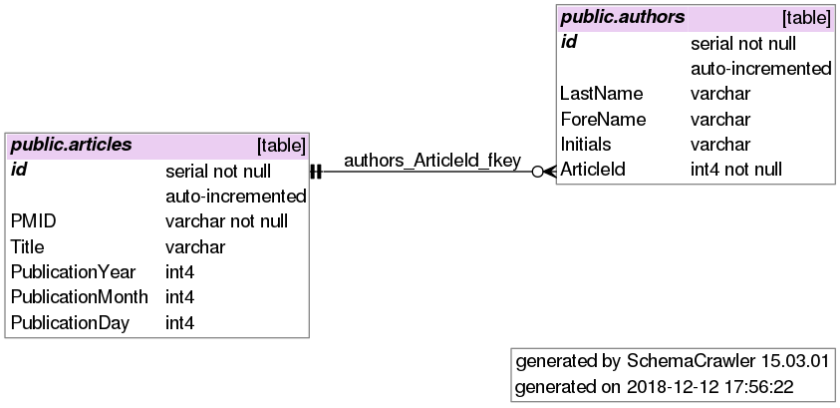

Figure 1: Entity relationship diagram of the database

1.2 Back-end

The assignment asked for a single page application, the back-end being there only to make the queries to the database, so I believe a simple *Node.js* API with the needed routes should do the work. I also used the *Sequelize* ORM to query the database.

1.3 Front-end

Since I have a really small experience with front-end frameworks, and since time was limited, I chose the one I read it had the more gentle learning curve, e.g. *Vue.js*. Without the time limiting constraint, I would probably have chosen *React* which has the biggest community today. I also used the *Vuetify* plugin to gain some time with already tuned components with nice CSS.

For the visualization, I chose to use the *Britecharts* library, which is built on top of *D3.js*, and which should require less time to learn how to use. Without the time limiting constraint, I would probably have chosen to go directly with *D3.js*.

I am aware the design doesn't exactly conform with all the requirements. I was struggling with time at the end so I had to make some compromises. With a little bit more time I could make it exactly as required.

2 Setup the app

Just run `docker-compose up` and open your browser at the following URL: `http://localhost:8080/`

3 Time spent

Table 1: Time spent on assignment

Design decisions	2h
Pulling data from PubMed API	2h
Database design and data parsing	6h
Back-end development	3h
Learning front-end framework	4h
Front-end development	8h
Docker containerization	4h

4 FDA 21 CFR 820.30

The FDA *21 CFR 820.30* relates to the development of software that might control a therapeutic device since it states that whatever devices automated with computer software are subject to design controls. These controls ensures that the device is safe, effective and traceable. They are about development planning, design input, design output, design review, design verification, design validation, design transfer, design changes. The FDA 22 CFR 820.30 also states that throughout the evolution of the device design, it has to maintain a design history file (DHF) to be able to track and justify every design step.

The implementation of *21 CFR 820.30* in the design and writing of software controlling a therapeutic device would be to first document the processes used to design and develop the software. The device requirements should also be documented, stating clearly what the device is supposed to do. This could mean writing the use cases. The process to get the build of the software produced has to be documented. There has to be formal procedures on how to review the written software, which could be peer reviewed for example, then we would have to state how a peer review has to be done. There has to be procedures to verify the software design. This could be a set of automatic tests made on each build of the software, running unit tests

and regression tests. The design has then to be validated. This could be a physical validation made by one or several beta testers following some test criteria. The software has then to be made ready for production. Containerization could be a solution to ensure the software is correctly translated to production. When the software changes over time, the design changes have to be documented. Every previous step have to be included in the DHF.